# A QPTAS for Gapless MEC

**Shilpa Garg**
Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
sgarg@mpi-inf.mpg.de
https://orcid.org/0000-0002-1825-0097

**Tobias Mömke**[1]
University of Bremen and Saarland University, Saarland Informatics Campus, Germany
moemke@cs.uni-saarland.de
https://orcid.org/0000-0002-2509-6972

## Abstract

We consider the problem Minimum Error Correction (MEC). A MEC instance is an $n \times m$ matrix $M$ with entries from $\{0, 1, -\}$. Feasible solutions are composed of two binary $m$-bit strings, together with an assignment of each row of $M$ to one of the two strings. The objective is to minimize the number of mismatches (errors) where the row has a value that differs from the assigned solution string. The symbol "$-$" is a wildcard that matches both 0 and 1. A MEC instance is gapless, if in each row of $M$ all binary entries are consecutive.

Gapless-MEC is a relevant problem in computational biology, and it is closely related to segmentation problems that were introduced by [Kleinberg–Papadimitriou–Raghavan STOC'98] in the context of data mining.

Without restrictions, it is known to be UG-hard to compute an $O(1)$-approximate solution to MEC. For both MEC and Gapless-MEC, the best polynomial time approximation algorithm has a logarithmic performance guarantee. We partially settle the approximation status of Gapless-MEC by providing a quasi-polynomial time approximation scheme (QPTAS). Additionally, for the relevant case where the binary part of a row is not contained in the binary part of another row, we provide a polynomial time approximation scheme (PTAS).

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis, Theory of computation → Dynamic programming

**Keywords and phrases** approximation algorithms, QPTAS, minimum error correction, segmentation, computational biology

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2018.34

**Acknowledgements** We would like to thank Tobias Marschall for helpful discussions.

## 1 Introduction

The minimum error correction problem (MEC) is a segmentation problem where we have to partition a set of length $m$ strings into two classes. A MEC instance is given by a set of $n$ strings over $\{0, 1, -\}$ of length $m$, where the symbol "$-$" is a wildcard symbol. The strings are represented by an $n \times m$ matrix $M$, where the $i$th string determines the $i$th row $M_{i,*}$ of $M$. The distance dist of two symbols $a, a'$ from $\{0, 1, -\}$ is $\text{dist}(a, a') := 1$ if $a = 0, a' = 1$ or $a = 1, a' = 0$ and $\text{dist}(a, a') := 0$ otherwise.

For two strings $s, s'$ from $\{0, 1, -\}^m$ where $s_j, s'_j$ denotes the $j$-th symbol of the respective string, $\text{dist}(s, s') := \sum_{j=1}^{m} \text{dist}(s_j, s'_j)$. A feasible solution to MEC is a pair of two strings

[1] Deutsche Forschungsgemeinschaft grant MO2889/1-1

$\sigma, \sigma'$ from $\{0,1\}^m$. The optimization goal is to find a feasible solution $(\sigma, \sigma')$ that minimizes $\text{cost}_M(\sigma, \sigma') := \sum_{i=1}^{n} \min\{\text{dist}(M_{i,*}, \sigma), \text{dist}(M_{i,*}, \sigma')\}$. If $M$ is clear from the context, we sometimes skip the index.

A MEC instance is called *gapless* if in each of the $n$ rows of $M$, all entries from $\{0,1\}$ are consecutive. (As regular expression, a valid row is a word of length $m$ from the language $-^*\{0,1\}^*-^*$). The MEC problem restricted to gapless instances is Gapless-MEC.

Our motivation to study Gapless-MEC stems from its applications in computational biology. Humans are diploid, and hence there exist two versions of each chromosome. Determining the DNA sequences of these two chromosomal copies – called haplotypes – is important for many applications ranging from population history to clinical questions [17, 18]. Many important biological phenomena such as compound heterozygosity, allele-specific events like DNA methylation or gene expression can only be studied when haplotype-resolved genomes are available [11].

Existing sequencing technologies cannot read a chromosome from start to end, but instead deliver small pieces of the sequences (called reads). Like in a jigsaw puzzle, the underlying genome sequences are reconstructed from the reads by finding the overlaps between them.

The upcoming next-generation sequencing technologies (e.g., Pacific Biosciences) have made the production of relatively long contiguous sequences with sequencing errors feasible, where the sequences come from both copies of chromosome. These sequences are aligned to a reference genome or to a structure called contig. We can formulate the result of this process as a Gapless-MEC instance: the sequences are the contiguous strings and the contig determines the columns of the strings.

Gapless-MEC is a generalization of a problem called Binary-MEC, the version of MEC with only instances $M$ where all entries of $M$ are in $\{0,1\}$. Finding an optimal solution to Binary-MEC is equivalent to solving the hypercube 2-segmentation problem (H2S) which was introduced by Kleinberg, Papadimitriou, and Raghavan [9, 10] and which is known to be NP-hard [4, 10]. The optimization version of Binary-MEC differs from H2S in that we minimize the number of mismatches instead of maximizing the number of matches. Binary-MEC allows for good approximations. Ostravsky and Rabiny [13] obtained a PTAS for Binary-MEC based on random embeddings. Building on the work of Li et al. [12], Jiao et al. [8] presented a deterministic PTAS for Binary-MEC.

Gapless-MEC was shown to be NP-hard by Cilibrasi et al. [3].[2] Additionally, they showed that allowing a single gap in each string renders the problem APX-hard. More recently, Bonizzoni et al. [2] showed that it is unique games hard to approximate MEC with constant performance guarantee, whereas it is approximable within a logarithmic factor in the size of the input. To our knowledge, previous to our result their logarithmic factor approximation was also the best known approximation algorithm for Gapless-MEC.

## 1.1 Our results

Our main result is the following theorem.

▶ **Theorem 1.** *There is a QPTAS for* Gapless-MEC.

Thus we partially settle the approximability for this problem: Gapless-MEC is not APX-hard unless $\mathsf{NP} \subseteq \mathsf{QP}$ (cf. [16]). Thus our result reveals a separation of the hardness of the gapless case and the case where we allow a single gap. Furthermore, already Binary-MEC is

[2] Their result predates the hardness result of Feige [4] for H2S. The proof of the claimed NP-hardness of H2S by Kleinberg, Papadimitriou, and Raghavan [9] was never published.

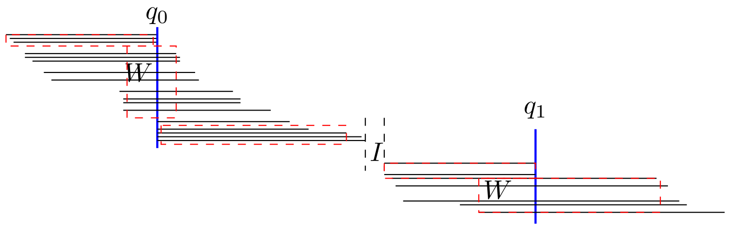

**Figure 1** Subinterval-free instance. Blocks represented by ranges shown in red on an instance $M$ and the blue lines are the columns, $I$ and $W$ shows the empty interval and central region respectively.

strongly NP-hard since the input does not contain numerical values. Therefore we can exclude the existence of an FPTAS for both Binary-MEC and Gapless-MEC unless P = NP.

Additionally, we address the class of *subinterval-free* Gapless-MEC instances where no string is contained in another string. More precisely, for each pair of rows from $M$ we exclude that the set of columns with binary entries from one row is a strict subset of the set of columns with binary entries from the other row.

▶ **Theorem 2.** *There is a PTAS for* Gapless-MEC *restricted to instances such that no string is the substring of another string.*

## 1.2 Overview of our approach

Our algorithm is a dynamic program (DP) that is composed of several levels. Given a general Gapless-MEC instance, we decompose the rows of the instance into length classes according to the length of the contiguous binary parts of the rows. For each length class we consider a well-selected set of columns such that each row crosses at least one column and at most two. (Row $i$ crosses a column $j$, if $M_{i,j} \in \{0, 1\}$.)

We decompose each length class into two sub-classes, one that crosses exactly one column and one that crosses exactly two columns. For the second class, it is sufficient to consider every other column, which leaves us with many *rooted* instances. Thus for each sub-instance there is a single column (the root) which is crossed by all rows of the instance.

We further decompose rooted sub-instances into the left hand side and the right hand side of the root. Since the two sides are symmetric, we can arrange the rows and columns of these sub-instances in such a way that all rows cross the first column. We call this type of sub-instance *SWC-instance* (for “simple wildcards”). We order the rows from top to bottom by increasing length in order to be able to further decompose the instance.

The first level of our DP solves these highly structured SWC-instances. The basic idea that we would like to apply is that we select a constant number of rows from the instance that represents the solution. Without further precautions, however, this strategy fails because of differing densities within the instance: the selected rows have to represent both the entries of columns crossed by many short rows and entries of arbitrarily small numbers of rows crossing many columns. To resolve this issue, we observe that computing the solution strings $\sigma$ and $\sigma'$ is equivalent to finding a partition of $M$ into two row sets, one assigned to $\sigma$ and the other assigned to $\sigma'$. If we assume to have the guarantee that for both solution strings $\sigma$ and $\sigma'$ an $\varepsilon$ fraction of rows of the matrix $M$ forms a Binary-MEC sub-instance, we show that the basic idea works.

This insight motivates to separate SWC-instances from left to right into sub-instances with the required property and to assemble them from left to right using a DP. There are, however, several complications. In order to choose the right sub-instances, we have to take into account that the choice depends on which rows are assigned to $\sigma$ and which are assigned to $\sigma'$. Therefore the DP has to take special care when identifying the sub-instances.

Furthermore, in order to stitch sub-instances together to form a common solution, the solution computed in the left sub-instance has to compute a set of candidate solutions oblivious of the choices of the right sub-instance. This means that we have to compute a solution to the left sub-instance without looking at a fraction of rows. We present an algorithm for these sub-instances in Section 2.

In order to combine the sub-instances, we face further technical complications due to having distinct sub-instances for those rows assigned to $\sigma$ and those rows assigned to $\sigma'$. In Section 2.1, we introduce a DP whose DP cells are pairs of simpler DP cells, one for $\sigma$ and one for $\sigma'$.

Before we consider general instances, in Section 3 develop our techniques by considering subinterval-free instances which are easier to handle (see Fig. 1). Observe that the instances considered until now are special rooted sub-interval-free instances. We show how to solve arbitrary rooted sub-interval-free instances by combining the DP with additional information about the sub-problems that contain the root. We then introduce the notion of domination in order to combine rooted sub-interval-free instances with a DP proceeding from left to right. The main idea is that a dominant sub-problem dictates the solution. At the interface of two sub-instances, there can be a (contiguous) region where none of the two sub-problems is dominant. We show that these regions can be solved directly by considering a constant number of rows (using the results from Section 2).

Until this point, all parts of our algorithm run in polynomial time. We lose this property when considering length classes, in Section 4.1. The length classes allow us to separate an instance into rooted sub-instances. The difficulty is that the left hand side of a separating column may have a completely different structure than the right hand side of that column. We do not know how to combine the two sides by considering only a polynomial number of possibilities. If we allow, however, quasipolynomial running time, we can solve the problem. We use that each of the two sub-instances (the one on the left and the one on the right) is composed of at most logarithmically many parts. Considering all parts simultaneously allows us to take care of dependencies between the left hand side and the right hand side and still solve them as if they were separate instances. Combining such rooted instances from left to right then can be done in the same spirit as combining rooted sub-interval-free instances. To solve the entire length-class, we combine both solutions by running a new DP that considers quadruples of DP cells.

Finally, in Section 4.2, we are able to handle all length classes simultaneously. We solve general instances in the same spirit as the combined sub-instances of a single length class. Instead of considering quadruples of cells, however, we form collections of quadruples that are – figuratively speaking – stacked on top of each other. The key insight is that there are only $O(\log(n))$ different length classes and each collection has at most one quadruple of each length class. Considering all possible collections adds another power of $\log(n)$ to the running time, which is still quasi-polynomial.

## 1.3 Further related work

Binary-MEC is a variant of the Hamming $k$-Median Clustering Problem when $k = 2$ and there are PTASs known [8, 13]. Li, Ma, and Wang [12] provided a PTAS for the general consensus pattern problem which is closely related to MEC. Additionally, they provided a PTAS for a restricted version of the star alignment problem aligning with at most a constant number of gaps in each sequence.

Alon and Sudakov [1] provided a PTAS for H2S, the maximization version of Binary-MEC and Wulff, Urner and Ben-David [19] showed that there is also a PTAS for the maximization version of MEC. For MEC, He et al. [7] studied the fixed-parameter tractability in the parameter of fragment length with some restrictions. These restrictions allow their dynamic programming algorithm to focus on the reconstruction of a single haplotype and, hence, to limit the possible combinations for each column. There is an FPT algorithm parameterized by the coverage [14, 6]. Bonizzoni et al. [2] provided FPT algorithms parameterized by the fragment length and the total number of corrections for MEC. There are some tools which can be used in practice to solve MEC instances [15, 14].

Most research in haplotype phasing deals with exact and heuristic approaches to solve MEC. Exact approaches, which solve the problem optimally, include integer linear programming [5] and fixed-parameter tractable algorithms [7, 15].

## 1.4 Preliminaries and notation

We consider a Gapless-MEC instance, which is a matrix $M \in \{0, 1, -\}^{n\times m}$. The $i$th row of $M$ is the vector $M_{i,*} \in \{0, 1, -\}^{1\times m}$ and the $j$th column is the vector $M_{*,j} \in \{0, 1, -\}^{n\times 1}$. The length of the binary part in $M_{i,*}$ is $|M_{i,*}|$. We say that the $i$th row of $M$ *crosses* the $j$th column if $M_{i,j} \in \{0, 1\}$.

For each feasible solution $(\sigma, \sigma')$ for $M$, we specify an assignment of rows $M_{i,*}$ to solution strings. The default assignment is specified as follows. For a row $M_{i,*}$, we assign $M_{i,*}$ to $\sigma$ if $\text{dist}(\sigma, M_{i,*}) \leq \text{dist}(\sigma', M_{i,*})$. Otherwise we assign $M_{i,*}$ to $\sigma'$. For the rows of $M$ assigned to $\sigma$ we write $\sigma(M)$ and for the rows assigned to $\sigma'$ we write $\sigma'(M)$. For a given instance, $\mathsf{Opt} = (\tau, \tau')$ denotes an optimal solution. Observe that knowing $\mathsf{Opt}$ allows us to obtain an optimal assignments $\tau(M)$ and $\tau'(M)$ by assigning each row to the solution string with fewest errors and knowing $\tau(M)$ and $\tau'(M)$ allows us to obtain an optimal solution by selecting the column-wise majority values.

## 2 Simple instances with wildcards

We consider instances of Gapless-MEC where all entries of column one in $M$ are zero or one, i.e., $M_{i,1} \in \{0, 1\}$ for each index $i$. Observe that the wildcards now have a simple structure which we refer to as SWC-structure. An instance with SWC-structure is an SWC-instance.

▶ **Definition 3** (Standard ordering of SWC-instances)**.** We define the *standard ordering* of rows in $M$ such that $|M_{i,*}| \leq |M_{i+1,*}|$ for each $i$, i.e., we order them from top to bottom in increasing length of the binary part.

▶ **Definition 4** (Good SWC-instances)**.** We call an SWC-instance $M$ *good*, if it is in standard ordering and there are at least $\varepsilon|\tau(M)|$ rows of $\tau(M)$ and at least $\varepsilon|\tau'(M)|$ rows of $\tau'(M)$ that have only entries from $\{0, 1\}$.

To solve good SWC-instances, we generalize the PTAS for Binary-MEC by Jiao et al. [8]. Our algorithm requires partitions of the set of rows. In the following two definitions, the required number of rows may be a fractional number. To solve the problem, we allow the assignment of fractional rows, i.e., for a row $i$, we can choose an $x \in [0, 1]$ and assign an $x$ fraction of $i$ to one set and a $1 - x$ fraction to the other set.

The following two definitions allow us to introduce a structured view on optimal solutions.

```
Algorithm 1: SWC$_\delta$.
Input  : Row sets $U_i$, $L_i$, $U'_i$ and $L'_i$ of a good SWC-instance $M$, numbers $r, r'$.
         Optional: selection of rows $\tilde{U}_i, \tilde{L}_i, \tilde{U}'_i, \tilde{L}'_i$, see below.
Output : A pair of solution strings $(\sigma, \sigma')$.
Run the algorithm for each possible selection of the following type and keep the best
  outcome (minimum number of errors);          // If provided as input, skip
  selection.
For each $i$, select (with repetition) a multi-set $\tilde{U}_i$ of $1/\delta$ rows from $U_i$ and $\tilde{L}_i$ from $L_i$;
For each $i$, select (with repetition) a multi-set $\tilde{U}'_i$ of $1/\delta$ rows from $U'_i$ and $\tilde{L}'_i$ from $L'_i$
  such that $\tilde{U}' \cap \tilde{U} = \tilde{L}' \cap \tilde{L} = \emptyset$;
// $\tilde{U} := \bigcup_i \tilde{U}_i$. The values $\tilde{U}'$, $\tilde{L}$, and $\tilde{L}'$ are defined analogously.
For each column $j$, set $\sigma_j := \textsc{Majority}_j(\tilde{U}, \tilde{L})$ and $\sigma'_j := \textsc{Majority}_j(\tilde{U}', \tilde{L}')$;
For each row $i$ of $M$, determine the value $d_i := \text{dist}(\sigma, M_{i,*}) - \text{dist}(\sigma', M_{i,*})$;
Assign the $r$ rows with minimal values $d_i$ to $\sigma$ and the remaining $r'$ rows to $\sigma'$.
```

▶ **Definition 5** (Trisection). An *$\varepsilon$-trisection* of an instance $M$ for $\tau$ is a partition of the rows into three consecutive ranges that have the following properties.

1. The first range $U$ contains row $M_{1,*}$ and $(1-\varepsilon)|\tau(M)|$ rows of $\tau(M)$.
2. The second range $L$ is consecutive to first row set containing $(\varepsilon - \varepsilon^2)|\tau(M)|$ rows of $\tau(M)$.
3. The third range $X$ contains the remaining rows in $M$.

To avoid ambiguity, we choose $L$ and $X$ such that the first row is in $\tau(M)$.

We define an $\varepsilon$-trisection $U'$, $L'$, and $X'$ for $\tau'$ analogously, replacing $\tau(M)$ by $\tau'(M)$.

▶ **Definition 6** (Subdivision of trisections). We consider the rows sets $U, L, U', L'$ from Definition 5 and additionally, we divide each of these sets into $1/\varepsilon^2$ disjoint subsets denoted as $U_i, L_i, U'_i, L'_i$. For each $i$, $U_i$ contains $\varepsilon^2 \cdot |U|$ rows from $\tau(M)$ and $L_i$ contains $\varepsilon^2 \cdot |L|$ rows from $\tau(M)$. Analogously, each $U'_i$ contains $\varepsilon^2 \cdot |U'|$ rows from $\tau'(M)$ and $L'_i$ contains $\varepsilon^2 \cdot |L'|$ rows from $\tau'(M)$. To avoid ambiguity, each set $U_i$ and $L_i$ starts with a (fractional) row of $\tau(M)$ and each set $U'_i$ and $L'_i$ starts with a (fractional) row of $\tau'(M)$.

We introduce a new algorithm SWC$_\delta$ for our setting. For an instance $M$, we consider the rows sets $U, L, U', L'$ from the $\varepsilon$-trisections of $M$ and their subsets according to Definition 6. Additionally, we select a multi-set of rows from $U'_i \cap \tau'(M)$ and $L'_i \cap \tau'(M)$. We then compute the majority weighting according to Definition 7 for each column $j$ using multisets based on the minimum number of errors. The main idea is to find two small row sets that represent the whole instance $M$. The intuitive meaning is that we select rows from the upper part with a much lower density then the rows of the lower part. We therefore introduce a bias such that all rows are equally important.

▶ **Definition 7** (Weighted majority). Let $j$ be an integer and let $\tilde{U}$ and $\tilde{L}$ be two matrices with at least $j$ columns. In $\tilde{U}_{*,j}$ and $\tilde{L}_{*,j}$, we replace all zeros by $-1$ and then all wildcard symbols by zero. We then compute the number $\nu := \sum_{i' \in \tilde{U}_{i,j}} (1-\varepsilon)i'/(\varepsilon - \varepsilon^2) + \sum_{i' \in \tilde{L}_{i,j}} i'$. Then $\textsc{Majority}_j(\tilde{U}, \tilde{L}) = 0$ if $\nu < 0$ and $\textsc{Majority}_j(\tilde{U}, \tilde{L}) = 1$ if $\nu \geq 0$.

With this preparation, we are now ready to present the algorithm. The input has a long list of parameters that will allow our dynamic programs later on to control the execution. The reason is that we do not know $\tau$ and $\tau'$. Therefore the algorithm takes *guesses* of row sets as input. The values $r$ and $r'$ are guesses of $|\tau(M)|$ and $|\tau'(M)|$.

Observe that for small (i.e., constant) values of $r$ or $r'$, the algorithm $\mathrm{SWC}_\delta$ can be replaced by an exact algorithm since we know $\tau(M)$ if and only if we know $\tau'(M)$, and we are able to guess constantly many rows.

▶ **Lemma 8.** *Let $M$ be a good SWC-instance. For sufficiently large $r = |\tau(M)|$ and $r' = |\tau'(M)|$, let $U_i, L_i, U_i', L_i'$ be a subdivision (Definition 6) of an $\varepsilon$-trisection $U, L, X, U', L', X'$ of $M$. Then $\mathrm{SWC}_{\varepsilon^3}$ is a $(1 + O(\varepsilon))$-approximation algorithm for $M$.*

The proof is based on a randomized argument using Chernoff bounds. In Lemma 8, we cannot control which rows of $X$ and $X'$ are assigned to which solution string. For our dynamic programs, we need a stronger statement. We would like to be able to compute a solution for an instance and *afterwards* change a fraction of assignments (guessing candidates for $\tau(X), \tau'(X')$) without losing the approximation guarantee. The next lemma is a key ingredient of our result.

▶ **Lemma 9.** *Let $M$ be a good SWC-instance and $\varepsilon > 0$ sufficiently small. Let $U, L, X$ be an $\varepsilon$-trisection for $\tau(M)$ and $U', L', X'$ an $\varepsilon$-trisection for $\tau'$, with subdivisions $U_i, L_i, U_i', L_i'$ according to Definition 6. Let $(\sigma, \sigma')$ be the solution computed by $\mathrm{SWC}_{\varepsilon^3}$ with $r = |\tau(M)|$, $r' = |\tau'(M)|$. Then re-assigning the rows $\sigma(X)$ to $\tau(X)$ and $\sigma'(X')$ to $\tau'(X')$ gives a $(1 + O(\varepsilon))$-approximation for the instance $M$.*

**Proof.** For ease of presentation, we assume that all appearing numbers are integers. It is easy to adapt the proof by rounding fractional numbers appropriately.

We first analyze the computed solution string $\sigma$. Let $\eta$ be the total number of errors of $(\tau, \tau')$ within $M$ and let $\eta_P$ be the total number of errors of $(\sigma, \sigma')$ within $P := U \cup L$. Due to Lemma 8, we have $\eta_P \le (1 + O(\varepsilon))\eta$.

We may assume $r \ge r'$ since otherwise we can simply rename the two strings $\tau$, $\tau'$. Additionally, by renaming of $\sigma$ and $\sigma'$, we may assume that $|\sigma(P) \cap \tau(P)| \ge |\sigma'(P) \cap \tau(P)|$. Therefore $|\tau(P)| \ge n/3$ and $|\sigma(P) \cap \tau(P)| \ge n/6$. (Recall that the matrix $M$ has $n$ rows and $m$ columns. The value $n/3$ is a safe bound on $n/2 - \varepsilon^2 n$, for $\varepsilon^2 \le 1/6$.)

▶ Claim 1. There is a set $I$ of $m - 25\eta/n$ indices $j$ such that $\sigma_j = \tau_j$ for all $j \in I$.

**Proof of Claim.** We concentrate on the columns of $M$ where both strings $\tau$ and $\sigma$ have at most $n/12$ errors within $P$. By counting the errors, there are at most $12\eta/n$ columns where $\tau$ has at least $n/12$ errors. Similarly, there are at most $12(1 + O(\varepsilon))\eta_P/n < 13\eta/n$ many columns where $\sigma$ has at least $n/12$ errors. Therefore there is a set $I$ of at least $m - 25\eta/n$ columns where simultaneously both $\tau$ and $\sigma$ have less than $n/12$ errors each.

Now suppose that the claim was not true and there was an index $j \in I$ with $\tau_j \ne \sigma_j$. Then, since $|\tau(P) \cap \sigma(P)| \ge n/6$, either $\sigma_j$ or $\tau_j$ is erroneous in at least $n/12$ rows of $\tau(P) \cap \sigma(P)$, a contradiction. ◊

Next we analyze $\sigma'$ for the columns $I$. Let $j$ be a column (i.e., an index) from $I$. By symmetry, we may assume $\sigma_j = \tau_j = 0$. We aim to show that an optimal solution has always sufficiently many errors to pay for wrong entries of $\sigma'$.

Let $\eta_j$ be the number of errors of $(\tau, \tau')$ in column $j$ of $M$ and let $\eta_{P,j}$ be the number of errors of $(\sigma, \sigma')$ in column $j$ of $P$. Let $\eta_j'' = \eta_j + \eta_{P,j}$.

▶ Claim 2. For each column $j$ of $I$, either $\sigma_j' = \tau_j'$ or $\eta_j'' \ge (\varepsilon - \varepsilon^2)|\tau'(M)|/2$.

**Proof of Claim.** We distinguish two cases. We first assume $\tau_j' = 0$. If also $\sigma_j' = 0$, we are done. We therefore assume $\sigma_j' = 1$. If there are more than $|\tau'(L')|/2$ ones in column $j$ of $L'$, $(\tau, \tau')$ has more than $|\tau'(L')|/2$ errors in column $j$ and thus $\eta_j \ge |\tau'(L')|/2$. Otherwise

$\sigma'(L')$ has at least $|\tau'(L')|/2$ zeros in column $j$ and therefore $\eta_{P,j} \geq |\tau'(L')|/2$. We obtain $\eta''_j \geq |\tau'(L')|/2 \geq (\varepsilon - \varepsilon^2)|\tau'(M)|/2$ as claimed.

In the second case, $\tau'_j = 1$ and we assume that $\sigma'_j = 0$. If there are more than $r'/2$ ones in column $j$ of $U'$, $(\sigma, \sigma')$ has more than $r'/2$ errors in column $j$ and thus $\eta_{P,j} \geq |\tau'(U')|/2$. Otherwise $\tau'(U')$ has at least $r'/2$ zeros in column $j$ and therefore $\eta_j \geq |\tau'(U')|/2$. Again, we obtain $\eta''_j \geq |\tau'(U')|/2 \geq (1-\varepsilon)|\tau'(M)|/2$ as claimed. ◊

Since by our assumption $|\tau'(X')| < \varepsilon^2|\tau'(M)|$, Claim 2 implies that within $I$, after reassigning the rows we still have a $(1 + O(\varepsilon))$-approximation.

To finish the proof, we argue that $\eta$ is large enough to pay for all errors in $X$ and $X'$ outside of $I$. Let $\eta_I$ be the number of errors due to assigning $\sigma$ to $\tau(X)$ and $\sigma'$ to $\tau'(X')$ within the interval $I$. Then, using the size of $I$ stated in Claim 1, the total number of errors of $(\sigma, \sigma')$ in $M$ is at most $(1 + O(\varepsilon))\eta + \eta_I + \varepsilon^2 n \cdot 25\eta/n$, i.e., the errors of $\mathrm{SWC}_{\varepsilon^3}$ within $P$, the errors within $X$ and $X'$ in the columns of $I$, and all other entries of $X \cup X'$. The obtained approximation ratio is $((1 + O(\varepsilon))\eta + \eta_I + \varepsilon^2 n \cdot 25\eta/n/\eta \leq (\eta + O(\varepsilon)\eta + 25\varepsilon^2\eta)/\eta = 1 + O(\varepsilon)$.

The first inequality uses that for some constant $k$, $(1 + k\varepsilon)\eta \geq \eta + \eta_I$. ◀

## 2.1 A DP for SWC-instances

Let $M$ be an SWC-instance with rows $\{1, 2, \ldots n\}$. We define $\text{start}_i$ to be the start and $\text{end}_i$ the end of string number $i$ of $M$, i.e., the column number of the matrix where the binary part starts and ends. For a sub-matrix $M'$ of $M$, $\text{start}_{M'}$ determines the index of the first column of $M'$ and $\text{end}_{M'}$ the index of the last column of $M'$. We next specify the parts of which the DP cells are composed. We divide the input instance into blocks defined as follows.

▶ **Definition 10** (Block). Given a good SWC-instance $M$, a block $B$ is a sub-instance determined by three numbers $1 \leq a < b < c \leq n$ as follows. The first column of $B$ is column 1 of $M$. The last column of $B$ is $\text{end}_b$. The first row of $B$ is $a$ and the last row is $n$. We write $U_B$ for the rows from $a$ to $b-1$, $L_B$ for the rows from $b$ to $c-1$, and $X_B$ for the rows from $c$ to $n$.

The idea is that a block determines a trisection. We subdivide each block into chunks and select rows from these chunks. Chunks are closely related to subdivisions of trisections, but we do not assume the knowledge of $(\tau, \tau')$.

▶ **Definition 11** (Chunk). Let $B$ be a block determined by the numbers $a, b, c$. We partition $B$ into $2/\varepsilon^2$ many *chunks* (ranges or rows). These chunks are determined by numbers $a = a_1 < a_2 < \cdots < a_{1/\varepsilon^2+1} = b = b_1 < b_2 < \cdots < b_{1/\varepsilon^2+1} = c$. The $\ell$th chunk of $U_B$ is the submatrix composed of the rows $a_\ell$ to $a_{\ell+1} - 1$ and the $\ell$th chunk of $L_B$ is the submatrix composed of the rows $b_\ell$ to $b_{\ell+1} - 1$.

▶ **Definition 12** (Selection). For each block $B$ with a set of chunks $C$, we consider multiset $T$ of rows of size $2/\varepsilon^5$. We require that $T$ contains $1/\varepsilon^3$ rows from each chunk in $C$.

The selection $T$ will take the role of $\tilde{U}$ and $\tilde{L}$ in $\mathrm{SWC}_\delta$.

▶ **Definition 13** (DP cell). For each block $B$, each set of chunks $C$ of $B$ and each selection $T$ of rows from $B$, there is a DP cell represented by $D(B, C, T)$. A DP cell $D(B, C, T)$ is a *predecessor* of $D(\hat{B}, \hat{C}, \hat{T})$ if the following conditions hold.

- $\hat{a} = b$ and $\hat{b} = c$, where $b, c, \hat{a}, \hat{b}$ are the numbers from Definition 10.
- The chunks from $C$ between $b$ and $c$ are exactly the chunks from $\hat{C}$ between $\hat{a}$ to $\hat{b}$.

- For each pair of chunks from $T \times \hat{T}$ with the same range of rows, the selections $T$ and $\hat{T}$ restricted to the pair are the same.

The value of $D(B, C, T)$ will be an approximation of the minimum number of errors that we can have in $M$ until the last column of $B$.

We now describe the dynamic program for a pair of solution strings $(\sigma, \sigma')$ by using joint DP cells $(\zeta, \zeta')$. For $\sigma'$, we use the same notation as in Definitions 10, 11 and 12, but we use the symbol prime ( $\cdot'$ ) for all occurring variables.

▶ **Definition 14** (DP cell for a pair)**.** A joint DP cell $(\zeta, \zeta') = (D(B, C, T),\ D'(B', C', T'))$ is composed of two single cells defined as in Definition 13. We require that

- the rows of $C$ and $C'$ where chunks start are pairwise distinct, and
- $T \cap T' = \emptyset$.

▶ **Definition 15** (Predecessor of a joint DP cell)**.** A DP cell $(\hat{\zeta}, \hat{\zeta}')$ is a *predecessor* of $(\zeta, \zeta')$ if (i) $\hat{\zeta} = \zeta$ and $\hat{\zeta}'$ is a predecessor of $\zeta'$; or (ii) $\hat{\zeta}$ is a predecessor of $\zeta$ and $\hat{\zeta}' = \zeta'$.

**Algorithm (SWC$^{\sigma,\sigma'}$).** The general idea of the algorithm is to guess trisections. Suppose we initially chose blocks $B, B'$ that are the trisections of the entire matrix $M$ for $\tau$ and $\tau'$. Then we obtain an approximation of the prefix of $(\tau, \tau')$ restricted to $B, B'$ (whichever ends first) by sampling rows of $U_B$,$L_B$,$U_{B'}$, and $L_{B'}$. The sampled rows for $L_B$ and $L_{B'}$ provide the interface to the next step. Suppose $L_{B'}$ starts at an earlier row than $L_B$. Then we guess the trisection of $M$ for $\tau$ restricted to the rows of $L_{B'}$ and $X_{B'}$. Let $B''$ be that block of our algorithm. Then $U_{B''} = L_B$ and we sample rows of $L_{B''}$ in order to approximate a new infix of $\tau$. More precisely, the DP does the following.

We globally guess a number $r$ that represent $|\tau(M)|$. Thus $r' := n - r$ represents $|\tau'(M)|$. We split the processing into an initialization phase and an update phase. In the initialization phase, we assign values to each DP cell $(\zeta, \zeta')$ based on SWC$_{\varepsilon^3}$ with the following parameters. We obtain $U_i, L_i$ from the chunks $C$ and $U'_i, L'_i$ from the chunks $C'$. In the execution of SWC$_{\varepsilon^3}$, we use the selections $T, T'$ instead of trying all possible selections, i.e., $T$ and $T'$ determine all $\tilde{U}_i$, $\tilde{L}_i$, $\tilde{U}'_i$, and $\tilde{L}'_i$ in the algorithm. Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c-1, c'-1\}$ and columns one to $\min\{\mathrm{end}_B, \mathrm{end}_{B'}\}$. The solution of the computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings until $\mathrm{end}_{\tilde{B}}$. The value of $(\zeta, \zeta')$ is $\mathrm{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$.

In the update phase, we compute the value and the pair of strings of the DP cell $(\zeta, \zeta')$ as follows. We inductively assume that all DP cells for predecessors of $(\zeta, \zeta')$ have been updated already. We try all predecessor pairs of DP cells and keep the one that gives the best result. Let $(\overline{\zeta}, \overline{\zeta}')$ be a predecessor of $(\zeta, \zeta')$. By symmetry, we assume without loss of generality that $b' < b$. There are two cases how the two pairs interact. The first case is $\zeta = \overline{\zeta}$. We run SWC$_{\varepsilon^3}$ on the columns $\mathrm{end}_{\overline{B}'} + 1$ to $\mathrm{end}_B$ with the parameters from $(\zeta, \zeta')$ (see initialization). To obtain the full solution, we append the computed string for $B'$ to the string $\sigma'_{\zeta,\overline{\zeta}'}$ (which is one of the solution strings of the predecessor pair). Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c - 1, c' - 1\}$ and columns one to $\mathrm{end}_{B'}$. The solution of the computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings from column one to $\mathrm{end}_{\tilde{B}}$. The potential new value of $(\zeta, \zeta')$ is $\mathrm{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$. We replace the stored solution with the potential new solution if the cost has decreased.

The second case is $\zeta' = \overline{\zeta}$. This case is the crux of the joint DP, since we have a "switch" of the role of $\sigma$ and $\sigma'$. We run SWC$_{\varepsilon^3}$ on the columns $\mathrm{end}_{\overline{B}}$ to $\mathrm{end}_{B'}$ with the parameters from $(\zeta, \zeta')$ (see initialization). To obtain the full solution, we then append the computed string for $B$ to the string $\sigma_{\overline{\zeta},\zeta'}$ (which is one of the solution strings of the predecessor pair). Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c-1, c'-1\}$ and columns one to $\text{end}_{B'}$. The solution of the computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings until $\text{end}_{\tilde{B}'}$. The potential new value of $(\zeta, \zeta')$ is $\text{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$. We replace the stored solution with the potential new solution if the cost has decreased.

For the last strings, we additionally consider special cells that are defined as before, but with $c = n$ or $c' = n$. Intuitively, we use these cells when only at most $1/\varepsilon^4$ rows of $\tau(M)$ or $\tau'(M)$ are left. For pairs of cells containing such $\zeta$ or $\zeta'$, our computation considers the optimal solution within the computation instead of $\text{SWC}_{\varepsilon^3}$.

▶ **Theorem 16.** *The algorithm* $\text{SWC}^{\sigma,\sigma'}$ *is a PTAS for SWC-instances.*

## 3 Subinterval-free instances

We show how to generalize the results of the previous section in order to handle instances where no interval of a string $s$ is a proper subinterval of a string $s'$ and thus show Theorem 2. To this end, we first show how to handle the rooted version of sub-interval free instances, where there is one column $j$ such that each string of the instance crosses $j$.

We order the rows of a subinterval-free instance $M$ from top to bottom such that for each pair $i, i'$ of rows with the binary part of $i$ starting on the left of the binary part of $i'$, $i$ is above $i'$. In other words, the binary strings are ordered from top to bottom with increasing starting position (i.e., column). Observe that the sub-string freeness property ensures that the last binary entry of $i'$ is not on the left of the last binary entry of $i$.

▶ **Lemma 17.** *Let $M$ be a* Gapless-MEC *instance such that no string is the substring of another string. Furthermore we assume that there is a column $j$ of $M$ such that each string of the instance crosses $j$. Then there is a PTAS for $M$.*

**General sub-interval-free instances.** We use Lemma 17 to handle general sub-interval free instances. Instead of a single column $j$ crossed by all strings, we determine a sequence $q = (q_1, q_2, \dots)$ of columns with the property that each string crosses exactly one of them. Let $s_1$ be the first string in $M$. Then we choose $q_1$ to be the column of the last entry of $s_1$.

We recursively specify the remaining columns. For a given $j$ such that we know $q_j$, let $s_i$ be the last (i.e., bottom-most) string that crosses $q_j$. Then we choose $q_{j+1}$ to be the last (i.e., rightmost) column of string $s_{i+1}$.

A simple induction shows that by the no-substring property and the chosen order of strings, each string crosses at least one column of $q$ and none of them crosses more than two. In particular, for each $j$, the solution on the left hand side of $q_j$ depends on rows of $M$ disjoint from the rows that determine the solution on the right hand side of $q_{j+1}$.

In order to combine the solution on the right hand side of $q_j$ with the solution on the left hand side of $q_{j+1}$, we introduce a notion of dominance.

▶ **Definition 18** (Dominance). We say that a submatrix $V_1$ of $M$ $\tau$-dominates a submatrix $V_2$ of $M$ if for each column $c$ that is in both $V_1$ and $V_2$, either at least one of the two matrices has no binary entries or the number of binary entries in $\tau(V_1)$ is at least $1/\varepsilon^2$ times the number in $\tau(V_2)$. We say that $V_1$ is $\tau$-dominant over $V_2$ for a column $c$, if the one column submatrix of $V_1$ determined by $c$ dominates $V_2$. We analogously define $\tau'$-dominance.

Consider a submatrix $\overrightarrow{V}$ of $M$ that only contains rows that cross $q_i$ and a submatrix $\overleftarrow{V}$ of $M$ that only contains rows that cross $q_{i+1}$. We observe that if $\overrightarrow{V}$ is $\tau$-dominant over $\overleftarrow{V}$

for some column $c$, it is also $\tau$-dominant for all columns on the left hand side of $c$: until $q_i$ is reached, when moving to the left the number of binary entries of $\tau(\overrightarrow{V})$ increases and the number of binary entries of $\tau(\overleftarrow{V})$ decreases. Analogously, if $\overleftarrow{V}$ is $\tau$-dominant over $\overrightarrow{V}$ for some column $c$, it is also $\tau$-dominant for all columns on the right hand side of $c$.

We therefore have a possibly empty interval $I$ without $\tau$-dominance such that the columns of $\overrightarrow{V}$ on the left hand side of $I$ are $\tau$-dominant and the columns of $\overleftarrow{V}$ on the right hand side of $I$ are $\tau$-dominant. (See also Figure 1.)

▶ **Definition 19** (Dominance region)**.** The *dominance region* of $\overrightarrow{V}$ with respect to $\overleftarrow{V}$ is the set of columns where $\overrightarrow{V}$ is dominant over $\overleftarrow{V}$, and vice versa.

Within the dominance region, our old DP can simply compute solutions without considering interferences: the dominated set of rows is small enough to be ignored, applying Lemma 9.

Within the interval $I$, the DP cells on both sides of $I$ have to "cooperate." We obtain a Binary-MEC block in the middle with additional rows on the top and bottom. This sub-instance can be solved directly.

## 4 A QPTAS for general instances

To solve the general instances, the main observation is that we divide the rows into their at most $\log_2(m)$ length classes $\Lambda_i$, and the $i$th length class $\Lambda_i$ is the set of all strings of length $\ell$ with $\ell \in (m/2^{i+1}, m/2^i]$. First we present an algorithm to solve each length class $\Lambda_i$ separately by constructing their corresponding columns.

### 4.1 Length classes

We show how we can handle length classes of strings. To this end, let us assume w.l.o.g. that $m$ (i.e., the number of columns in $M$) is a power of 2. Then for each $i \geq 0$, the $i$th length class $\Lambda_i$ is the set of all strings of length $\ell$ with $\ell \in (m/2^{i+1}, m/2^i]$. We observe the following known property of length classes.

▶ **Lemma 20.** *For each $i \geq 0$ there is a set $q_i = \{q_{i,1}, q_{i,2}, \dots\}$ of columns such that (a) each string in $\Lambda_i$ crosses at least one column from $q_i$ and (b) no string from $\Lambda_i$ crosses more than two columns from $q_i$. Furthermore, we can choose the sets such that $q_i \subseteq q_{i+1}$.*

**Proof.** At level $i$, for each $k$ with $1 \leq k \leq 2^{i+1}$ we select the column with index $k \cdot m/2^{i+1}$. We observe that the distance between two consecutive columns from $q_i$ is $m/2^{i+1}$, which matches the shortest length of strings in $\Lambda_i$: if a minimal string starts right after a column of $q_i$, its last entry will cross the next column of $q_i$.

Since strings do not start before column 1 and column $m$ is contained in each $q_i$, claim (a) follows. To see (b), observe that a maximum length string of $\Lambda_i$ is at most $m/2^i$. Let $j$ be an index. The number of columns from $q_{i,j}$ to the column right before $q_{i,j+1}$ and from $q_{i,j+1}$ to right before $q_{i+2}$ are exactly $m/2^{i+1}$ . If the string starts directly at a column $q_{i,j}$ from $q_i$, it would cross column $q_{i,j+1}$ and end right before column $q_{i,j+2}$. The last claimed property follows directly from the construction of the sets $q_i$. ◀

For each $i$, we now separate $\Lambda_i$ into two sub-instances. One sub-instance $\Lambda_i'$ is formed by those rows from $\Lambda_i$ that only cross one column of $q_i$ and the second sub-instance $\Lambda_i''$ is formed by those rows that cross exactly two columns of $\Lambda_i$.

▶ **Definition 21** (DP for a length class $\Lambda_i$)**.** For each index $j$ let $\xi'_j$ be the sets of DP cells for $\Lambda'_i$ and for the odd indices $j$ let $\xi''_j$ be the set of cells for $\Lambda''_i$. We define a super-cell that starts in $j$, $(Z'_j, Z''_j, Z'_{j+1}, Z''_{j+2}) \in \xi'_j \times \xi''_j \times \xi'_{j+1} \times \xi''_{j+2}$ and the super-cell that ends in $j$, $(Z'_{j-1}, Z''_{j-2}, Z'_j, Z''_j) \in \xi'_{j-1} \times \xi''_{j-2} \times \xi'_j \times \xi''_j$.

▶ **Lemma 22.** *There is a QPTAS for* Gapless-MEC *if all strings are in the same class* $\Lambda_i$.

## 4.2 The general QPTAS

Finally we combine our insights to an algorithm for general instances by combining different length classes. For different length classes $\Lambda_i$, we construct their corresponding columns as explained in the previous section. The main idea is that for each column $j$, we only have to consider those quadruple of super-cells according to Definition 21 that cross $j$ from all the length classes simultaneously. We therefore consider at most $O(\log(n))$ quadruples of super-cells simultaneously. In the dynamic program, we consider a joint quadruple of super-cells from all the length classes. Then the overall complexity of a joint cell is quasi-polynomial: the number of different cells is $\left(n^{O(\log n)}\right)^{O(\log n)} = n^{O(\log^2 n)}$.

Let $Q_{i,j}$ be the set of quadruples of length class $i$ crossing column $j$ such that the strings are ordered from shortest length class to the longest. For each length class $i$, a quadruple $q \in Q_{i,j}$ is the set of rows starting at $j$, cross $j$, or end in $j$. If $j$ is the index of $q_{i,\ell}$, the quadruple $q$ starts in $j$ if it is formed by cells $(Z'_\ell, Z''_\ell, Z'_{\ell+1}, Z''_{\ell+2})$ and ends in $j$ if it is formed by $(Z'_{\ell-1}, Z''_{\ell-2}, Z'_\ell, Z''_\ell)$ (see Definition 21). If $j$ lies between $q_{i,\ell}$ and $q_{i,\ell+1}$, $j$ crosses those quadruples that contain $Z'_\ell$ and $Z'_{\ell+1}$. If none of the cases are true, we do not consider $q$ in the cells for column $j$.

Let us consider a $\log(n)$ vector of quadruples $v$, with one quadruple $Q_{i,j}$ for each $i$ and, consider quadruples starting at, ending at, or crossing column $j$ for length class $i$. We require that if for some $i$, the quadruple $q \in Q_{i,j}$ ends at $j$, then for all the length classes $\Lambda_k$ with $k > i$ the same condition holds (with index larger than $i$). This also implies that if for some $i$, the quadruple of length class $i$ starts at $j$, then the same also holds for all quadruples of shorter length classes (with index larger than $i$). In particular, in order to be able to combine neighboring vectors of quadruples, we do not allow to mix starting and ending quadruples. Let $\phi$ be the set of all $\log(n)$ vectors of tuples as described above (with one tuple of each length class). The tuple for each length class is defined as in Lemma 22 and the DP for general instances follows the ideas of Lemma 22: We move from left to right column by column. In the initialization step, the joint DP cell is initialized based on Algorithm 1 using $\phi$. We guess the blocks, chunks and selections from each length class and consider them jointly in a DP cell.

For column $j$, let us consider a vector $v \in \phi$. We distinguish whether $v$ has starting or ending quadruples. (One of the two cases must apply due to the shortest length class.) For a $v \in \phi$ with starting quadruples, let $d$ be the smallest number such that there is a quadruple of length class $d$ starting at $j$. To compute $v$ we consider all $v' \in \phi$ with the following properties. (a) $v'$ has the same quadruples for all length classes $d' < d$ and (b) for $d' \geq d$, the right hand sides of the quadruples of length class $d'$ in $v'$ compatible the left hand sides of the quadruples of $v$. The super-cells from the left and right hand side are compatible if the intersecting strings from the left and right hand side are assigned to the same types of solution string $\sigma$ or $\sigma'$.

For a $v \in \phi$ with ending quadruples, let $d$ be the smallest number such that there is a quadruple of length class $d$ ending at $j - 1$. (In the very first column of the instance, we do not need this value.) To compute $v$ we consider all $v' \in \phi$ with the following properties. (a)

$v'$ has the same quadruples for all length classes $d' < d$ and (b) for $d' \geq d$, the right hand sides of the quadruples of length class $d'$ in $v'$ match the left hand sides of the quadruples of $v$ in column $j-1$. Then the value of $v$ is the sum of the minimum value over all such $v'$ and the number of errors in column $j$ obtained by applying $\mathrm{SWC}_{\varepsilon^3}$ exactly as in the proof of Lemma 22.

The approximation ratio follows by arguing that the expected number of errors at each column is at most $(1+O(\varepsilon))$ of OPT (see Lemma 22). This finishes the proof of Theorem 1.

## References

1 Noga Alon and Benny Sudakov. On two segmentation problems. *Journal of Algorithms*, 33(1):173–184, 1999.

2 Paola Bonizzoni, Riccardo Dondi, Gunnar W Klau, Yuri Pirola, Nadia Pisanti, and Simone Zaccaria. On the minimum error correction problem for haplotype assembly in diploid and polyploid genomes. *Journal of Computational Biology*, 2016.

3 Rudi Cilibrasi, Leo van Iersel, Steven Kelk, and John Tromp. The Complexity of the Single Individual SNP Haplotyping Problem. *Algorithmica*, 49(1):13–36, aug 2007. `doi:10.1007/s00453-007-0029-z`.

4 Uriel Feige. NP-hardness of hypercube 2-segmentation. *CoRR*, abs/1411.0821, 2014.

5 Pierre Fouilhoux and A. Ridha Mahjoub. Solving VLSI design and DNA sequencing problems using bipartization of graphs. *Computational Optimization and Applications*, 51(2):749–781, 2012. `doi:10.1007/s10589-010-9355-1`.

6 Shilpa Garg, Marcel Martin, and Tobias Marschall. Read-based phasing of related individuals. *Bioinformatics*, 32(12):i234–i242, 2016.

7 Dan He, Arthur Choi, Knot Pipatsrisawat, Adnan Darwiche, and Eleazar Eskin. Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 26(12):i183–i190, 2010. `doi:10.1093/bioinformatics/btq215`.

8 Yishan Jiao, Jingyi Xu, and Ming Li. On the $k$-closest substring and $k$-consensus pattern problems. In *CPM*, volume 3109 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2004.

9 Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. Segmentation problems. In *STOC*, pages 473–482. ACM, 1998.

10 Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. Segmentation problems. *J. ACM*, 51(2):263–280, 2004.

11 Danny Leung, Inkyung Jung, Nisha Rajagopal, Anthony Schmitt, Siddarth Selvaraj, Ah Young Lee, Chia-An Yen, Shin Lin, Yiing Lin, Yunjiang Qiu, et al. Integrative analysis of haplotype-resolved epigenomes across human tissues. *Nature*, 518(7539):350–354, 2015.

12 Ming Li, Bin Ma, and Lusheng Wang. Finding similar regions in many sequences. *J. Comput. Syst. Sci.*, 65(1):73–96, 2002.

13 Rafail Ostrovsky and Yuval Rabani. Polynomial-time approximation schemes for geometric min-sum median clustering. *J. ACM*, 49(2):139–156, 2002.

14 Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo van Iersel, Leen Stougie, Gunnar W. Klau, and Alexander Schönhuth. WhatsHap: Weighted haplotype assembly for future-generation sequencing reads. *Journal of Computational Biology*, 22(6):498–509, feb 2015. `doi:10.1089/cmb.2014.0157`.

15 Yuri Pirola, Simone Zaccaria, Riccardo Dondi, Gunnar W. Klau, Nadia Pisanti, and Paola Bonizzoni. HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*, page btv495, aug 2015. `doi:10.1093/bioinformatics/btv495`.

16 Jan Remy and Angelika Steger. Approximation schemes for node-weighted geometric steiner tree problems. *Algorithmica*, 55(1):240–267, 2009.

17 Matthew W Snyder, Andrew Adey, Jacob O Kitzman, and Jay Shendure. Haplotype-resolved genome sequencing: experimental methods and applications. *Nature Reviews Genetics*, 16(6):344–358, 2015.
18 Ryan Tewhey, Vikas Bansal, Ali Torkamani, Eric J Topol, and Nicholas J Schork. The importance of phase information for human genomics. *Nature Reviews Genetics*, 12(3):215–223, 2011.
19 Sharon Wulff, Ruth Urner, and Shai Ben-David. Monochromatic bi-clustering. In *ICML (2)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 145–153. JMLR.org, 2013.